

Quantum Ternary Circuit Synthesis Using Projection Operations

SUDHINDU BIKASH MANDAL^{1*}, AMLAN CHAKRABARTI¹,
SUSMITA SUR-KOLAY²

¹ A. K. Choudhury School of Information Technology, University of Calcutta, India

² Advanced Computing & Microelectronics Unit, Indian Statistical Institute, India

Basic logic gates and their operations in ternary quantum domain are involved in the synthesis of ternary quantum circuits. Only a few works define ternary algebra for ternary quantum logic realization. In this paper, a ternary logic function is expressed in terms of projection operations including a new one. A method to realize new multi-qutrit ternary gates in terms of generalized ternary gates and projection operations is also presented. We also introduced ten simplification rules for reducing ancilla qutrits and gate levels. Our method yields lower gate cost and fewer gate levels and ancilla qutrits than that obtained by earlier methods for the ternary benchmark circuits. The n qutrit ternary sum function is synthesized without any ancilla qutrit by our proposed methodology.

Key words: Gate level logic synthesis, Ternary quantum circuits, Projection operations

1 INTRODUCTION

Multi-valued quantum computing is gaining importance in the field of quantum information theory and quantum cryptography as it can represent an n -dimensional quantum system, defined by the basis states $|0\rangle$, $|1\rangle$, ...,

* email: sudhindu.mandal@gmail.com

$|n - 1 \rangle$. The unit of information is called a qudit. A qudit exists as a linear superposition of states, and is characterized by a wave function ψ [1, 2].

Multi-valued quantum algebra comprises the rules for a set of basic logic operations that can be performed on qudits. While in [3] the structure of a multi-valued logic gate is proposed which can be experimentally feasible with a linear ion trap scheme for quantum computing, this approach produces large dimensional circuits. A universal architecture for multi-valued reversible logic is given in [4], but quantum realization of the circuits thus obtained is not apparent. The universality of n -qudit gates is presented in [5], but no algorithms for synthesis were given. Al-Rabedi proposed in [6] the minimization technique for multi-valued quantum galois field sum of product.

In this paper we concentrate on the quantum ternary functions. A ternary quantum system exists in linear superposition of three basis states, labeled $|0 \rangle$, $|1 \rangle$ and $|2 \rangle$. All the operations on a qutrit are developed in a 3-dimensional Hilbert space under the field GF(3) [7]. A two qutrit vector can be represented as $|\psi \rangle = \sum_{i=0}^2 |C_i|^2 = 1$. In general, a n qutrit state can be represented as a superposition of 3^n basis states. A quantum register of size n qutrits can hold 3^n values simultaneously, whereas the n -qubit register in binary quantum domain can hold 2^n values. For the same size of memory, a Quantum Fourier Transform (QFT) with ternary qutrits improves the approximation and increases the state space by a factor of $(3/2)^n$ [8]. This is very useful in cryptography.

The realization of ternary operations requires a set of universal gates with which a quantum circuit for a given ternary quantum function can be built. One set of universal gates [9] comprise the Ternary Swap, the Ternary NOT and the Ternary Toffoli gate. With these universal gates, any arbitrary quantum circuit can be realized without ancilla bits, but this approach is not optimal in terms of quantum cost [9]. Further, the synthesis method in [10] uses quantum multiplexers and the method of iterative deepening depth first search is employed to minimize the gate cost. Another synthesis technique was proposed in [11] for garbage free GF(3) based reversible or quantum logic circuit from its truth values, using only Muthukrishnan-Stroud (M-S) [3] and shift gates [12]. But this technique did not provide any simplification rule to reduce the gate count. In paper [13] the ternary quantum logic was expressed in terms of Ternary Galois Field Sum of Product, and 16 Ternary Galois Field expansions were also proposed. But this paper did not provide any gate level implementation of ternary benchmark circuit.

The paper here is the extended version of [14], where our motivation

is to synthesis a given quantum ternary benchmark [13] function specified as ternary minterms, using projection operations. Further we realize the gate level implementation of these benchmark [13] circuits using Generalized Ternary Gates [8] and newly defined *permutative* ternary quantum C²NOT gates. A few simplification rules are enumerated for reducing the number of ancilla qutrits and the levels of gates in the resulting quantum circuit.

The preliminary concepts of ternary algebra with a new projection operation appear in Section 2. In Section 3, the characteristics of basic ternary logic gates and the Generalized Ternary Gate (GTG) [12] along with the introduction of a few new gates are discussed. The proposed synthesis methodology along with a set of simplification rules is presented in Section 4. Synthesis of quantum ternary benchmark circuits by this method, and the respective quantum gate cost with ancilla qutrit count are given in Section 5. Concluding remarks appear in Section 6.

2 TERNARY ALGEBRA

We first define the basic operations over the set $\{0, 1, 2\}$.

2.1 Ternary AND, OR, NOT

The operations of AND, OR, and NOT [15] for ternary variables are:

$$AND(a, b) = \begin{cases} a & \text{if } a \leq b; \\ b & \text{otherwise.} \end{cases} \quad OR(a, b) = \begin{cases} a & \text{if } a \geq b; \\ b & \text{otherwise.} \end{cases}$$

$$NOT(a) = a + 1.$$

Here '+' denotes the modulo 3 addition. It is easy to see that the ternary AND operation is distributive over the OR operation and vice-versa [15].

2.2 Ternary Projection Operations L_i and J_i

Our main goal is to define a given ternary function in terms of the minterms corresponding to its 1 or 2 values. The 3^n minterms for a function f with n ternary variables are denoted by m_0 to m_{3^n-1} . In the sum of products form of minterms, the minterm $m_0 = \prod_i^n NOT(x_i)$ where $x = \{0, 1, 2\}$ and \prod denotes the ternary AND; its value is 1, 2, or 0 depending on whether for all i $x_i = 0, 1$ or 2 respectively.

As the outcomes of a ternary function may be 0, 1 or 2, we cannot express a ternary function f which results in either 1 or 2 for a given set of inputs, by using the above minterms m_i . So, we present six projection operations, grouped into two types L_i and J_i , where $i = \{0, 1, 2\}$. While the J_i type operation was defined earlier [15] as

a	$L_0(a)$	$L_1(a)$	$L_2(a)$	$J_0(a)$	$J_1(a)$	$J_2(a)$
0	1	0	0	2	0	0
1	0	1	0	0	2	0
2	0	0	1	0	0	2

TABLE 1
Truth tables of ternary projection operations L_i and J_i

$$J_i(a) = \begin{cases} 2 & \text{if } a = i; \\ 0 & \text{otherwise.} \end{cases}$$

the L_i types are newly defined here as

$$L_i(a) = \begin{cases} 1 & \text{if } a = i; \\ 0 & \text{otherwise.} \end{cases}$$

Their truth tables appear in Table 1. From Table 1, we can verify that the output of any L_i operation is either 0 or 1, and that of J_i operation is either 0 or 2. Two other derived operations L'_i, J'_i as shown in the Table 2. Next, we express a ternary function as a sum of products minterm form by using the L_i and J_i operations. The L_i and J_i operations are commutative, associative and distributive over ternary AND and OR logic. We can verify these laws given below, from Table 1. Here $i, j, k = \{0, 1, 2\}$

1. Commutativity

- (i) $L_i(a) + L_j(b) = L_j(b) + L_i(a)$
 $J_i(a) + J_j(b) = J_j(b) + J_i(a)$
- (ii) $L_i(a).L_j(b) = L_j(b).L_i(a)$
 $J_i(a).J_j(b) = J_j(b).J_i(a)$

2. Associativity

- (i) $[L_i(a) + L_j(b)] + L_k(c) = L_i(a) + [L_j(b) + L_k(c)]$
 $[J_i(a) + J_j(b)] + J_k(c) = J_i(a) + [J_j(b) + J_k(c)]$
- (ii) $[L_i(a).L_j(b)].L_k(c) = L_i(a).[L_j(b).L_k(c)]$
 $[J_i(a).J_j(b)].J_k(c) = J_i(a).[J_j(b).J_k(c)]$

3. Distributivity

- (i) $L_i(a).[L_j(b) + L_k(c)] = L_i(a).L_j(b) + L_i(a).L_k(c)$
 $J_i(a).[J_j(b) + J_k(c)] = J_i(a).J_j(b) + J_i(a).J_k(c)$
- (ii) $L_i(a) + [L_j(b) + L_k(c)] = [L_i(a) + L_j(b)].[L_i(a) + L_k(c)]$
 $J_i(a) + [J_j(b) + J_k(c)] = [J_i(a) + J_j(b)].[J_i(a) + J_k(c)]$

a	$L'_0(a)$	$L'_1(a)$	$L'_2(a)$	$J'_0(a)$	$J'_1(a)$	$J'_2(a)$
0	0	1	1	0	2	2
1	1	0	1	2	0	2
2	1	1	0	2	2	0

TABLE 2
Truth tables of ternary operations L'_i and J'_i

3 TERNARY LOGIC GATES

3.1 Ternary Feynman, Ternary Toffoli, MAX and MIN gates

The two qutrit ternary Feynman gate [16], shown in Figure 1, is defined as

$$\text{Feynman}(A, B) = A + B$$

where '+' operator is addition over GF(3).

The ternary 3-qutrit Toffoli gate [16] in the universal ternary quantum gate set, is shown in Figure 2 and is defined as

$$\text{Toffoli}(A, B, C) = \begin{cases} Z \text{ operation on } C & \text{if } A = B = 2; \\ C & \text{otherwise;} \end{cases}$$

where A, B are controlling input and $Z = \{+1, +2, 01, 02, 12\}$.

In our synthesis method, ternary SWAP and NOT gates are not used explicitly. Instead, we use the ternary MAX and MIN gates [10] respectively to replace the OR and the AND in ternary quantum logic. These two gates are defined as

$$\text{MAX}(A_1, A_2, \dots, A_n, B) = \begin{cases} A_i & \text{if } A_i \geq A_j, i \neq j \text{ and } A_i \geq B; \\ B & \text{if } \forall i, B \geq A_i; \end{cases}$$

$$\text{MIN}(A_1, A_2, \dots, A_n, B) = \begin{cases} A_i & \text{if } A_i \leq A_j, i \neq j \text{ and } A_i \leq B; \\ B & \text{if } \forall i, B \leq A_i; \end{cases}$$

where $i = \{1, 2, \dots, n\}$ (Figure 3).

The ternary Toffoli gate can be realized without any ancilla qutrits [16, 17] by using Muthukrishnan-Stroud gates (M-S gate) and . An M-S gate is defined as

$$\text{M-S}(A, B) = \begin{cases} B \text{ shift by } Z & \text{if } A = 2; \\ B & \text{otherwise;} \end{cases}$$

where $Z = \{+1, +2, 01, 02, 12\}$ [15] (Figure 4(a)).

3.2 Generalized Ternary Gate

In order to realize a Generalized ternary gate (GTG) defined next, the Shift gates [12] given in Table 3 are required. Here addition and multiplication

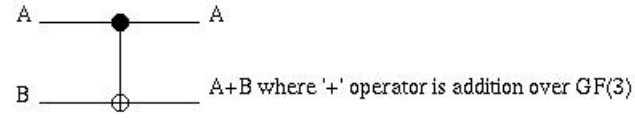


FIGURE 1
2 qutrit ternary Feynman Gate

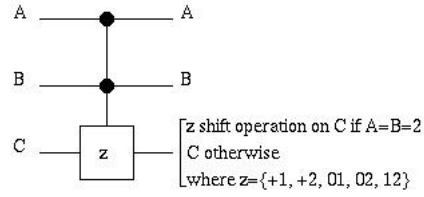


FIGURE 2
3 qutrit ternary Toffoli Gate

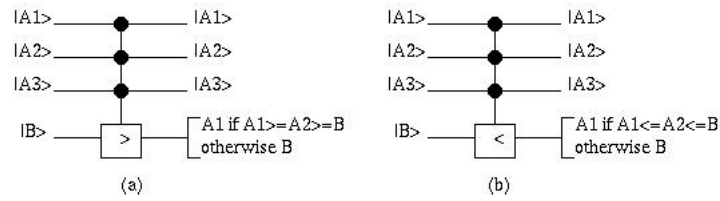


FIGURE 3
Ternary Multi-qutrit (a) *MAX* gate, and (b) *MIN* gate

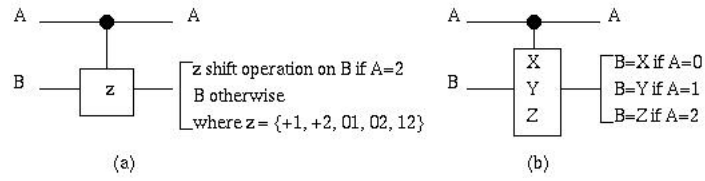


FIGURE 4
(a) A Muthukrishnan-Stroud (M-S) gate, and (b) a Generalized Ternary Gate (GTG)

TABLE 3
Ternary Shift operations used in a GTG

Shift gate	Its operation
Buffer	$x = x$
Single Shift	$x = x + 1$
Dual Shift	$x = x + 2$
Self Shift	$x = 2x$
Self Single Shift	$x = 2x + 1$
Self Dual Shift	$x = 2x + 2$

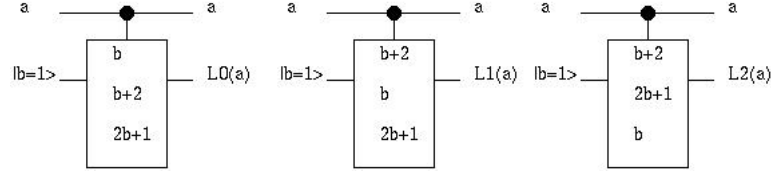


FIGURE 5
Realization of ternary (a) L_i operation and (b) J_i operation with GTGs

over $GF(3)$ are denoted by '+' and '.' respectively.

A Generalized Ternary Gate (GTG) is a 2-qutrit gate, as shown in Figure 4(b). The controlling input of a GTG can be used to select the 1-qutrit shift operation on the target input. The GTG is defined as

$$GTG(A, B) = \begin{cases} B \text{ shift } X & \text{if } A=0; \\ B \text{ shift } Y & \text{if } A=1; \\ B \text{ shift } Z & \text{if } A=2 \end{cases}$$

where X , Y , and Z are any three distinct shift operations in Table 3.

Implementation of L_i and J_i operations using GTG

We can implement the L_i and J_i projection operations defined in Section 2.2, by using GTG as shown in Figures 5 (a) and (b) respectively. For the L_i type operation, we set $b = 1$.

$$L_i(a) = GTG(a, b) = \begin{cases} b & \text{if } a=i; \\ b+2 & \text{if } a=i+1; \\ 2b+1 & \text{if } a=i+2 \end{cases}$$

For the J_i type operation, we set $b = 2$.

$$J_i(a) = GTG(a, b) = \begin{cases} b & \text{if } a=i; \\ b+1 & \text{if } a=i+1; \\ 2b+2 & \text{if } a=i+2 \end{cases}$$

3.3 A new Ternary C^2NOT gate

Several multi-qutrit control operations are possible in ternary logic. We present a new definition of a 3-qutrit C^2NOT (Figure 6(a)), which is required to realize the simplification rules for ternary minterms given in the Section 4.2 below, as

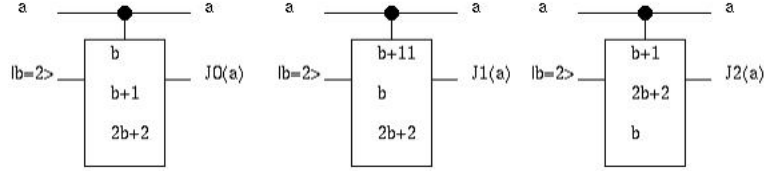


FIGURE 6
Realization of ternary J_i operation with GTG

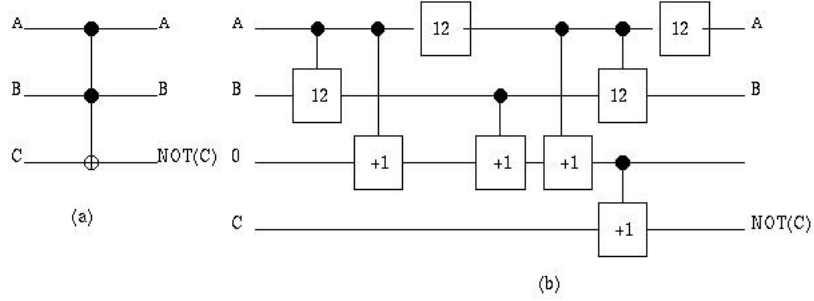


FIGURE 7
(a) The proposed ternary C^2NOT gate, and (b) its realization with M-S gates

$$C^2NOT(A, B, C) = \begin{cases} NOT(C) & \text{if } A \neq B \text{ and } A, B \neq 0; \\ C & \text{otherwise;} \end{cases}$$

where A and B are the control inputs, and C the target input. The realization of this C^2NOT gate with M-S gates is shown in Figure 6(b).

3.4 A Multi-qutrit GTG

We also define a Multi-qutrit GTG (Figure 7) as

$$GTG(A_1, A_2, \dots, A_n, B) = \begin{cases} B \text{ shift X} & \text{if } A_1, A_2, \dots, A_n = 0; \\ B \text{ shift Y} & \text{if } A_1, A_2, \dots, A_n = 1; \\ B \text{ shift Z} & \text{if } A_1, A_2, \dots, A_n = 2; \\ B & \text{otherwise;} \end{cases}$$

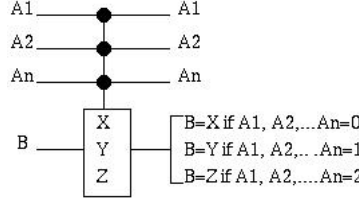


FIGURE 8
The proposed Multi-qutrit GTG

4 PROPOSED SYNTHESIS METHODOLOGY

4.1 Overview

Consider an m -variable ternary quantum logic function $f(a_1, a_2, \dots, a_m) = \sum_{i=0}^n (\text{minterms for one})_i + \sum_{j=0}^p (\text{minterms for two})_j$ where \sum implies logical ternary OR, n and p are respectively the number of input vectors for which f is 1, and 2. Thus, for $(3^m - n - p)$ input vectors, f is 0. We express the minterm for *one* and *two* by using the L_i and J_i operations respectively. From Table II, we can say

$$\begin{aligned} \prod_{i=1}^m L_0(a_i) &= \begin{cases} 1 & \text{if } \forall i \ a_i = 0; \\ 0 & \text{if } \exists i \ a_i = 1 \text{ or } 2; \end{cases} \\ \prod_{i=1}^m L_1(a_i) &= \begin{cases} 1 & \text{if } \forall i \ a_i = 1; \\ 0 & \text{if } \exists i \ a_i = 0 \text{ or } 2; \end{cases} \\ \prod_{i=1}^m L_2(a_i) &= \begin{cases} 1 & \text{if } \forall i \ a_i = 2; \\ 0 & \text{if } \exists i \ a_i = 0 \text{ or } 1; \end{cases} \\ \prod_{i,p,k=1}^m L_0(a_i) \cdot L_1(a_p) \cdot L_2(a_k) &= \begin{cases} 1 & \text{if } \forall i, p, k \ a_i = 0, a_p = 1, a_k = 2; \\ 0 & \text{if } \exists i, p, k \ a_i = 1 \text{ or } 2, a_p = 0 \text{ or } 2, a_k = 0 \text{ or } 1; \end{cases} \end{aligned}$$

where $i + p + k = m$.

Hence, the minterms for which $f = 1$ are

1. $\prod_{i=1}^m L_0(a_i = 0)$,
2. $\prod_{i=1}^m L_1(a_i = 1)$,
3. $\prod_{i=1}^m L_2(a_i = 2)$, and
4. $\prod_{i,p,k=1}^m L_0(a_i = 0) \cdot L_1(a_p = 1) \cdot L_2(a_k = 2)$.

Similarly from Table I, the minterms for which $f = 2$ are

1. $\prod_{i=1}^m J_0(a_i = 0)$,
2. $\prod_{i=1}^m J_1(a_i = 1)$,
3. $\prod_{i=1}^m J_2(a_i = 2)$, and

$$4. \prod_{i,p,k=1}^m J_0(a_i = 0) \cdot J_1(a_p = 1) \cdot J_2(a_k = 2).$$

4.2 Simplification Rules

Next, we define seven simplification rules for our proposed method, the first four are derived from Table 1, and the next three from Table 2.

1. $L_i(a) \cdot 0 = 0$, and $J_i(a) \cdot 0 = 0$
2. $L_i(a) \cdot 1 = L_i(a)$, and $J_i(a) \cdot 2 = J_i(a)$
3. $L_i(a) + 0 = L_i(a)$, and $J_i(a) + 0 = J_i(a)$
4. $L_i(a) + 1 = 1$, and $J_i(a) + 2 = 2$
5. $L_i(a) \cdot L'_i(a) = 0$, and $J_i(a) \cdot J'_i(a) = 0$
6. $L_i(a) + L'_i(a) = 1$, and $J_i(a) + J'_i(a) = 2$
7. $L'_i(a) = L_{i+1}(a) + L_{i+2}(a)$, and $J'_i(a) = J_{i+1}(a) + J_{i+2}(a)$.

Simplification Rules for reducing ancilla qutrits

For gate level realization of the projection operations L_i and J_i , we need an ancilla qutrit for each. Further, to synthesize an m -variable ternary function with n minterms specified according to our proposed methodology, we have maximum of $n * m$ ancilla qutrits. However, we can reduce the number of ancilla qutrits by the following three simplification rules based on the new ternary C^2NOT gate and Table 2:

8. $L_1(a)L_2(b)+L_2(a)L_1(b) = C^2NOT(a, b, 0)$ and $J_1(a)J_2(b)+J_2(a)J_1(b) = C^2NOT(a, b, 1)$
9. $L_i(a) \cdot L_i(a) = L_i(a)$ and $J_i(a) \cdot J_i(a) = J_i(a)$
10. $L_i(a_1)L_i(a_2) \dots L_i(a_n) = L_i(a_1, a_2, \dots, a_n)$ and $J_i(a_1)J_i(a_2) \dots J_i(a_n) = J_i(a_1, a_2, \dots, a_n)$, $i = \{0, 1, 2\}$.

The multiquitrit L_i and J_i operations in Rule 10 can be realized using Multiquitrit GTG as follows:

$$\text{For } L_i(a_1, a_2, \dots, a_n), GTG(a_1, a_2, \dots, a_n, B) = \begin{cases} B+1 & \text{if } a_1, a_2, \dots, a_n=i; \\ B & \text{if } a_1, a_2, \dots, a_n=i+1; \\ 2B & \text{if } a_1, a_2, \dots, a_n=i+2; \\ B & \text{otherwise;} \end{cases}$$

$$\text{For } J_i(a_1, a_2, \dots, a_n), GTG(a_1, a_2, \dots, a_n, B) = \begin{cases} B+2 & \text{if } a_1, a_2, \dots, a_n=i; \\ B & \text{if } a_1, a_2, \dots, a_n=i+1; \\ 2B & \text{if } a_1, a_2, \dots, a_n=i+2; \\ B & \text{otherwise;} \end{cases}$$

4.3 An Example

For illustrating our method, consider an arbitrary ternary function $g(a, b)$ with its truth table given in Table 4.

TABLE 4

Truth table for an example ternary 2-qutrit function $g(a, b)$

a, b	00	01	02	10	11	12	20	21	22
$g(a, b)$	0	1	2	1	1	1	2	1	2

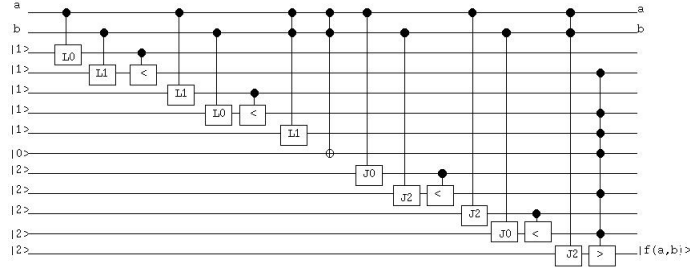


FIGURE 9

Ternary quantum gate circuit obtained for $g(a, b)$ in Table 4

The disjunction of minterms for which $g = 1$ is

$$L_0(a)L_1(b) + L_1(a)L_0(b) + L_1(a)L_1(b) + L_1(a)L_2(b) + L_2(a)L_1(b)$$

and that for which $g = 2$ is

$$J_0(a)J_2(b) + J_2(a)J_0(b) + J_2(a)J_2(b).$$

Hence, $g(a, b) = L_0(a)L_1(b) + L_1(a)L_0(b) + L_1(a)L_1(b) + L_1(a)L_2(b) + L_2(a)L_1(b) + J_0(a)J_2(b) + J_2(a)J_0(b) + J_2(a)J_2(b)$.

By simplification rules 1 and 10, we get

$$g(a, b) = L_0(a)L_1(b) + L_1(a)L_0(b) + L_1(a, b) + C^2 NOT(a, b, 0) + J_0(a)J_2(b) + J_2(a)J_0(b) + J_2(a, b).$$

The gate level implementation of the function $g(a, b)$ is shown in Figure 8.

5 SYNTHESIS OF TERNARY BENCHMARK FUNCTIONS

We experimented with the ternary benchmark functions provided in [13]. Their truth tables appear in Table 5. The definition and the synthesis of the ternary benchmark functions using ternary minterm are given in the following

subsections.

5.1 2-qutrit multiplication functions *mul2* and *mul2c*

The functions are defined as

$mul2(a, b) = ab \bmod 3$, multiplication output;

$mul2c(a, b) = \text{int}[ab/3]$, carry of the multiplier.

The truth table of $mul2(a, b)$ and $mul2c(a, b)$ appear in columns 3 and 4 respectively of Table 5.

Expressing in terms of projectin operators, we get

$$mul2(a, b) = L_1(a)L_1(b) + L_2(a)L_2(b) + J_1(a)J_2(b) + J_2(a)J_1(b).$$

$$= L_1(a, b) + L_2(a, b) + C^2 NOT(a, b, 1) \text{ (by Rules 8 and 10)}$$

Similarly, $mul2c(a, b) = L_2(a)L_2(b) = L_2(a, b)$ (by Rule 10).

5.2 2-qutrit half-adder *thadd*

The 2-qutrit half-adder *thadd* [14] comprises the ternary half adder sum (*sumh*) and carry (*carryh*) functions, whose truth tables are given in columns 5 and 6 respectively of Table 5.

We re-write *sumh* according to our proposed methodology as

$$sumh = L_0(a)L_1(b) + L_1(a)L_0(b) + L_2(a)L_2(b) + J_0(a)J_2(b) + J_1(a)J_1(b) + J_2(a)J_0(b).$$

Further simplification of *sumh* by using the ternary Feynman gate of Section 3.1 gives $sumh = a + b$.

Next, the carry function *carryh* is written as

$$carryh = L_1(a)L_2(b) + L_2(a)L_1(b) + L_2(a)L_2(b) = C^2 NOT(a, b, 0) + L_2(a, b) \text{ (by Rules 8 and 10)}.$$

5.3 GF(3) sum of two squares *sqsum2*

The definition of this ternary function is $sqsum2(a, b) = (a^2 + b^2) \bmod 3$ and its truth table is shown in column 7 of Table 5.

$$\begin{aligned} sqsum2(a, b) &= L_0(a)L_1(b) + L_0(a)L_2(b) + L_1(a)L_0(b) + L_2(a)L_0(b) + \\ &\quad J_1(a)J_1(b) + J_1(a)J_2(b) + J_2(a)J_1(b) + J_2(a)J_2(b) \\ &= L_0(a)L'_0(b) + L'_0(a)L_0(b) + J_1(a, b) + C^2 NOT(a, b, 2) + J_2(a, b) \\ &\quad \text{(by Rules 8, 7 and 10).} \end{aligned}$$

5.4 Average function *avg2*

The function *avg2* is the integer part of the average of two ternary input variables in *modulo3* and defined as $avg2(a, b) = \text{int}[ab/2]$. Its truth table is in column 8 of Table 5.

$$avg2 = L_0(a)L_2(b) + L_1(a)L_1(b) + L_1(a)L_2(b) + L_2(b)L_0(b) + L_2(a)L_1(b) + J_2(a)J_2(b)$$

TABLE 5
Truth table of four 2-qutrit benchmark functions[13]

a	b	$mul2$	$mul2c$	$sumh$	$carryh$	$sqsum2$	$avg2$
0	0	0	0	0	0	0	0
0	1	0	0	1	0	1	0
0	2	0	0	2	0	1	1
1	0	0	0	1	0	1	0
1	1	1	0	2	0	2	1
1	2	2	0	0	1	2	1
2	0	0	0	2	0	1	1
2	1	2	0	0	1	2	1
2	2	1	1	1	1	2	2

$$=L_0(a)L_2(b)+L_1(a,b)+C^2NOT(a,b,1)+L_2(a)L_0(b)+J_2(a,b)$$

(by Rules 8 and 10).

5.5 3-qutrit multiplication function $mul3$

The function $mul3$ is defined as

$mul3(a,b,c) = abcm3mod3$, multiplication output;

$mul3c(a,b) = int[abc/3]$, carry of the multiplier.

From the truth table of $mul3(a,b,c)$ in column 4 of Table 6, we have

$$\begin{aligned}
 mul3(a,b,c) &= L_1(a)L_1(b)L_1(c)+L_1(a)L_2(b)L_2(c)+L_2(a)L_1(b)L_2(c)+ \\
 &\quad L_2(a)L_2(b)L_1(c)+J_1(a)J_1(b)J_2(c)+J_1(a)J_2(b)J_1(c)+ \\
 &\quad J_2(a)J_1(b)J_1(c)+J_2(a)J_2(b)J_2(c) \\
 &= L_1(a)[L_1(b)L_1(c)+L_2(b)L_2(c)]+L_2(a)[L_1(b)L_2(c)+ \\
 &\quad L_2(b)L_1(c)]+J_1(a)[J_1(b)J_2(c)+J_2(b)J_2(c)]+J_2(a)[J_1(b)J_1(c)+ \\
 &\quad J_2(b)J_2(c)]
 \end{aligned}$$

By applying L_i and J_i operations on $mul2$ in column 3 of Table 5, we get

$$\begin{aligned}
 &=L_1(a)L_1(mul2(b,c))+L_2(a)L_2(mul2(b,c))+J_1(a)J_2(mul2(b,c))+ \\
 &\quad J_2(a)J_1(mul2(b,c)). \\
 &= mul2(a, mul2(b,c)).
 \end{aligned}$$

From the truth table of $mul3c(a,b,c)$ in column 5 of Table 6, we have

$$\begin{aligned}
 mul3c(a,b,c) &= L_1(a)L_2(b)L_2(c)+L_2(a)L_1(b)L_2(c)+L_2(a)L_2(b)L_1(c)+ \\
 &\quad J_2(a)J_1(b)J_2(c) \\
 &= C^2NOT(a,b,0)L_2(c)+L_2(a,b)L_1(c)+J_2(a,b,c) \text{ by} \\
 &\quad \text{Rules 8 and 10).}
 \end{aligned}$$

TABLE 6
Truth table of four 3-qutrit benchmark functions [13]

a	b	c	$mul3$	$mul3c$	$a2bcc$	$avg3$	$sqsum3$
0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	1
0	0	2	0	0	2	0	1
0	1	0	0	0	0	0	1
0	1	1	0	0	2	0	2
0	1	2	0	0	1	1	2
0	2	0	0	0	0	0	1
0	2	1	0	0	0	1	2
0	2	2	0	0	0	1	2
1	0	0	0	0	1	0	1
1	0	1	0	0	2	0	2
1	0	2	0	0	0	1	2
1	1	0	0	0	1	0	2
1	1	1	1	0	0	1	0
1	1	2	2	0	1	1	0
1	2	0	0	0	1	1	2
1	2	1	2	0	1	1	0
1	2	2	1	1	1	1	0
2	0	0	0	0	1	0	1
2	0	1	0	0	2	1	2
2	0	2	0	0	0	1	2
2	1	0	0	0	1	1	2
2	1	1	2	0	0	1	0
2	1	2	1	1	2	1	0
2	2	0	0	0	1	1	2
2	2	1	1	1	1	1	0
2	2	2	2	2	1	2	0

5.6 Function a^2bcc

This is an arbitrary function defined as $a^2bcc = (a^2 + bc + c) \bmod 3$ with its truth table shown in column 6 of Table 6. Hence,

$$\begin{aligned}
 a^2bcc &= L_0(a)L_0(b)L_1(c) + L_0(a)L_1(b)L_2(c) + L_1(a)L_0(b)L_0(c) + L_1(a)L_1(b)L_0(c) + \\
 &\quad L_1(a)L_1(b)L_2(c) + L_1(a)L_2(b)L_0(c) + L_1(a)L_2(b)L_1(c) + L_1(a)L_2(b)L_2(c) + \\
 &\quad L_2(a)L_0(b)L_0(c) + L_2(a)L_1(b)L_0(c) + L_2(a)L_2(b)L_0(c) + L_2(a)L_2(b)L_1(c) + \\
 &\quad L_2(a)L_2(b)L_2(c) + J_0(a)J_0(b)J_2(c) + J_0(a)J_1(b)J_1(c) + J_1(a)J_0(b)J_1(c) + \\
 &\quad J_2(a)J_0(b)J_1(c) + J_2(a)J_1(b)J_2(c) \\
 &= L_0(a, b)L_1(c) + L_0(a)L_1(b)L_2(c) + L'_0(a)L_0(b, c) + L_1(a, b)L'_1(c) + \\
 &\quad C^2NOT(a, b, 0)L_0(c) + L_1(a, c)L_2(b) + L_1(a)L_2(b, c) + L_2(a, b)L'_2(c) + \\
 &\quad L_2(a, b, c) + J_0(a, b)J_2(c) + J_0(a)J_1(a, c) + J_1(a, c)J_0(b) + J_2(a)J_0(b)J_1(c) + \\
 &\quad J_2(a, c)J_1(b) \text{ (by Rules 7, 8, 10).}
 \end{aligned}$$

5.7 Function $avg3$ function

This function is the integer part of the average of three ternary input variables expressed as *modulo3*, i.e., $avg3 = [int[a + b + c/3]] \bmod 3$. Its truth table is shown in column 7 of Table 6.

$$\begin{aligned}
 avg3 &= L_0(a)L_1(b)L_2(c) + L_0(a)L_2(b)L_1(c) + L_0(a)L_2(b)L_2(c) + L_1(a)L_0(b)L_2(c) + \\
 &\quad L_1(a)L_1(b)L_1(c) + L_1(a)L_1(b)L_2(c) + L_1(a)L_2(b)L_0(c) + L_1(a)L_2(b)L_1(c) + \\
 &\quad L_1(a)L_2(b)L_2(c) + L_2(a)L_0(b)L_1(c) + L_2(a)L_0(b)L_2(c) + L_2(a)L_1(b)L_0(c) + \\
 &\quad L_2(a)L_1(b)L_1(c) + L_2(a)L_1(b)L_2(c) + L_2(a)L_2(b)L_0(c) + L_2(a)L_2(b)L_1(c) + \\
 &\quad J_2(a)J_2(b)J_2(c). \\
 &= L_0(a)C^2NOT(b, c, 0) + L_0(a)L_2(b, c) + L_0(b)C^2NOT(a, c, 0) + L_1(a, b, c) + \\
 &\quad L_1(a)C^2NOT(b, c, 0) + L_0(c)C^2NOT(a, b, 0) + L_2(c)C^2NOT(a, b, 0) + L_0(b) \\
 &\quad L_2(a, c) + L_2(a)L_1(b, c) + L_2(a, b)L_0(c) + L_2(a, b)L_1(c) + J_2(a, b, c) \\
 &\text{(by Rules 8 and 10)} \\
 &= L'_2(a)C^2NOT(b, c, 0) + L_0(a)L_2(b, c) + L_0(b)C^2NOT(a, c, 0) + L_1(a, b, c) + \\
 &\quad L'_1(c)C^2NOT(a, b, 0) + L_0(b)L_2(a, c) + L_2(a)L_1(b, c) + L_2(a, b)L'_2(c) + J_2(a, b, c) \\
 &\text{(by Rule 7).}
 \end{aligned}$$

5.8 GF(3) sum of three squares $sqsum3$

The definition is $sqsum3(a, b, c) = (a^2 + b^2 + c^2) \bmod 3$ with the truth table in column 8 of Table 6.

$$\begin{aligned}
 sqsum3(a, b) &= L_0(a)L_0(b)L_1(c) + L_0(a)L_0(b)L_2(c) + L_0(a)L_1(b)L_0(c) + L_0(a)L_2(b)L_0(c) + \\
 &\quad L_1(a)L_0(b)L_0(c) + L_2(a)L_0(b)L_0(c) + J_0(a)J_1(b)J_1(c) + J_0(a)J_1(b)J_2(c) + \\
 &\quad J_0(a)J_2(b)J_1(c) + J_0(a)J_2(b)J_2(c) + J_1(a)J_0(b)J_1(c) + J_1(a)J_0(b)J_2(c) + \\
 &\quad J_1(a)J_1(b)J_0(c) + J_1(a)J_2(b)J_0(c) + J_2(a)J_0(b)J_1(c) + J_2(a)J_0(b)J_2(c) + \\
 &\quad J_2(a)J_1(b)J_0(c) + J_2(a)J_2(b)J_0(c) \\
 &= L_0(a, b)L_1(c) + L_0(a, b)L_2(c) + L_0(a, c)L_1(b) + L_0(a, c)L_2(b) +
 \end{aligned}$$

$$\begin{aligned}
& L_1(a)L_0(b,c)+L_2(a)L_0(b,c)+J_0(a)J_1(b,c)+J_0(a)C^2NOT(b,c,1)+ \\
& J_0(a)J_2(b,c)+J_1(a,c)J_0(b)+J_0(b)C^2NOT(a,c,1)+J_1(a,b)J_0(c)+ \\
& C^2NOT(a,b,1)J_0(c)+J_2(a,c)J_0(b)+J_2(a,b)J_0(c) \text{ (by Rules} \\
& 8 \text{ and } 10) \\
& = L_0(a,b)L'_0(c)+L_0(a,c)L'_0(b)+L'_0(a)L_0(b,c)+J_0(a)J_1(b,c)+J_0(a) \\
& C^2NOT(b,c,1)+J_0(a)J_2(b,c)+J_1(a,c)J_0(b)+J_0(b)C^2NOT(a,c,1) \\
& +J_1(a,b)J_0(c)+C^2NOT(a,b,1)J_0(c)+J_2(a,c)J_0(b)+J_2(a,b)J_0(c) \\
& \text{(by Rule 9)}
\end{aligned}$$

5.9 GF(3) product *prod4*

The function *prod4* is defined as $prod4(a,b,c,d) = (abcd)mod3$.

$$\begin{aligned}
prod4(a,b,c,d) &= L_1(a)L_1(b)L_1(c)L_1(d)+L_1(a)L_1(b)L_2(c)L_2(d)+L_1(a)L_2(b)L_1(c)L_2(d)+ \\
& L_1(a)L_2(b)L_2(c)L_1(d)+L_2(a)L_1(b)L_1(c)L_2(d)+L_2(a)L_1(b)L_2(c)L_1(d)+ \\
& L_2(a)L_2(b)L_1(c)L_1(d)+L_2(a)L_2(b)L_2(c)L_2(d)+J_1(a)J_1(b)J_1(c)J_2(d)+ \\
& J_1(a)J_1(b)J_1(c)J_1(d)+J_1(a)J_2(b)J_2(c)J_2(d)+J_2(a)J_1(b)J_1(c)J_1(d)+ \\
& J_2(a)J_1(b)J_2(c)J_2(d)+J_2(a)J_2(b)J_1(c)J_2(d)+J_2(a)J_2(b)J_2(c)J_1(d)+ \\
& J_1(a)J_1(b)J_2(c)J_1(d) \\
& = mul2(mul2(a,b),mul2(c,d)) \text{ (by Rules 8, 10 and pro-} \\
& \text{jection operation on } mul2 \text{ in column 3 of Table 5).}
\end{aligned}$$

5.10 4-qutrit GF(3) addition function *sum4*

This function is $sum4(a,b,c,d) = (a+b+c+d)mod3$. Along the lines of the result for ternary full adder in [14], we have

by applying the simplification rules 8, 10 and projection operation on *sumh* in column 5 of Table 5, the minterms of *sum4* as

$$\begin{aligned}
sum4(a,b,c,d) &= L_0(a+b)L_1(c+d) + L_1(a+b)L_0(c+d) + L_2(a+b) \\
& L_2(c+d) + J_0(a+b)J_2(c+d) + J_1(a+b)J_1(c+d) + J_2(a+b)J_0(c+d) \\
& = sumh(sumh(a,b),sumh(c,d)).
\end{aligned}$$

5.11 Synthesis Results

We synthesized the ternary benchmark circuit provided in [13] using our proposed synthesis methodology. The simplified forms of these functions by our method appear in Section 5.1 to 5.10 and the costs are summarized in Table 7. While the second column of the table indicates the maximum number of ancilla qutrits that may be required to synthesize the benchmark circuits, the third column shows the number of ancilla qutrit required after using our simplification rules in Section 4.2.

The fourth column shows the M-S gate count cost. The quantum gate cost of the ternary quantum benchmark circuits are evaluated in terms of the number of M-S gates used. The number of M-S gates required to realize a

Feynman gate, a Toffoli and a GTG gate are 4, 5 and 5 respectively [11, 18]. To realize the new ternary C^2NOT gate shown in Figure 7b, we need 8 M-S gates.

For the ternary *sumn* function, the number of ancilla qutrits is zero and the quantum gate cost is $(n - 1) * 4$. In the case of ternary *prodn* function, the maximum number of ancilla qutrits that may be required is $2^n * n$. But by applying our simplification rules, we need only $(n - 1) * 3$ ancilla qutrits for the realization of *prodn* function, with a quantum gate cost of $(n - 1) * 18$ for this function.

Our proposed simplification rules reduce more than 50% of the ancilla qutrits for the functions *mul2*, *mul3*, *avgn* and *sqsumn*. The comparison of the M-S gate count cost with that in [19] for these benchmark circuits are given in Table 7. This establishes that although our design has a small increase in cost for the *sqsumn* and *avgn* circuits, it is 20% less for all the other benchmark circuits.

6 CONCLUSION

In this paper, we have proposed a methodology for logic synthesis of ternary quantum circuits. We have defined a minterm based approach of expressing a ternary logic function by using L_i and J_i operations. We have also stated the simplification rules for the proposed method. By applying these simplification rules, we can realize the ancilla free *sumn* function. Among the ternary benchmark circuits [13], the quantum gate cost for the group of addition and multiplication functions *sumn* and *prodn*, by our method is smaller compared to that by the earlier method in [19].

REFERENCES

- [1] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, 2002.
- [2] A. Ekert and A. Zeilinger, *The physics of quantum information*, Springer Verlag, Berlin, 2002, pp. 1-14.
- [3] A. Muthukrishnan and C. R. Stroud Jr., *Multi-Valued Logic Gates for Quantum Computation*, Phys. Rev. A62, 2000, pp. 0523091-8.
- [4] P. Picton, *A Universal Architecture for Multiple-Valued Reversible Logic*, Multiple-Valued Logic - An International Journal, Vol. 5, 2000, pp. 27-37
- [5] J. L. Brylinski and R. Brylinski, *Universal Quantum Gates*, (to appear in Mathematics of Quantum Computation, CRC Press, 2002) LANL e-print quant-ph/010862..

TABLE 7
Quantum Gate cost of Ternary Benchmark [13] Circuit

Circuit Name	Maximum An-cilla qutrit	Reduced Ancilla qutrit	Gate Cost	
			our	[19]
<i>sum2</i>	12	0	4	5
<i>sum3</i>	54	0	8	10
<i>sum4</i>	216	0	12	15
<i>sum5</i>	810	0	16	-
<i>sum6</i>	2916	0	20	-
<i>sum7</i>	10206	0	24	-
<i>prod2</i>	8	3	18	20
<i>prod3</i>	24	6	36	65
<i>prod4</i>	64	9	54	-
<i>prod5</i>	160	12	72	-
<i>prod6</i>	384	15	90	-
<i>prod7</i>	896	18	108	-
<i>mul2</i>	10	4	23	25
<i>mul3</i>	36	11	64	-
<i>thadd</i>	18	2	21	20
<i>tfadd</i>	63	4	42	55
<i>avg2</i>	12	7	38	15
<i>avg3</i>	51	16	89	40
<i>sqsum2</i>	16	7	38	10
<i>sqsum3</i>	54	24	130	15

- [6] M. Perkowski, A. Al-Rabadi, P. Kerntopf, A. Mishchenko, and M. Chrzanowska-Jeske, *Three-Dimensional Realization of Multivalued Functions Using Reversible Logic*, Booklet of 10th Int. Workshop on Post-Binary Ultra-Large-Scale Integration Systems (ULSI), Warsaw, Poland, May 2001, pp.47- 53
- [7] A. B. Klimov, L. Luis, S. Soto, H. Guise and G. Björk, *Quantum phase of a qutrit*, J. Phys. A: Math. Gen. 37, 2004, pp. 4097-4103.
- [8] Z. Zilic and K. Radecka, *Scaling and Better Approximating Quantum Fourier Transform by Higher Radices*, IEEE Transaction on Computers, Vol. 56, 2007, pp. 202-207.
- [9] G. Yang, X. Song, M. Perkowski, J. Wu, *Realizing Ternary Quantum Switching Networks without Ancilla Bits*, J. Phys. A: Math. Gen., 2005, pp. 9689-9697.
- [10] N. Giesecke, D. H. Kim, S. Hossain and M. Perkowski, *Search for Universal Ternary Quantum Gate Sets with Exact Minimum Costs*, Proceedings of RM Symposium, Oslo, May

16, 2007.

- [11] M. M. Khan, A. K. Biswas, S. Chowdhury, M. Hasan, A. I. Khan *Synthesis of $GF(3)$ Based Reversible/Quantum Logic Circuits Without Garbage Output*, 39th International Symposium on Multiple-Valued Logic, Okinawaw, Japan, 2009, pp. 98-102.
- [12] M. Perkowski, A. Al-Rabadi, and P. Kerntopf, *Multiple-Valued Quantum Logic Synthesis*, International Symposium on New Paradigm VLSI Computing, Sendai, Japan, 2002, pp. 41-47.
- [13] M. H. A. Khan, M. Perkowski, and M. R. Khan, *Ternary Galois Field Expansions for Reversible Logic and Kronecker Decision Diagrams for Ternary GFSOP Minimization*, 34th International Symposium on Multiple-Valued Logic, 2004, pp. 41-47.
- [14] S.B.Mandal, A. Chakrabarti, and S. Sur-Kolay, *Synthesis Technique for Ternary Quantum Logic*, 41 th International Symposium on Multiple-Valued Logic, Tuusula, 2011, pp. 218-223.
- [15] R. L. Herrmann, *Selection and implementation of a ternary switching algebra*, spring joint computer conference, Atlantic City, 1968, pp. 283-290 .
- [16] M. H. A. Khan, *Design of Reversible Quantum Ternary Multiplexer and Demultiplexer*, Engineering Letters, 13:2, 2002, pp. 65-69.
- [17] M. H. A. Khan, *Quantum Realization of Multiple-Valued Feynman and Toffoli Gates without Ancilla Input*, 39th International Symposium on Multiple-Valued Logic, Okinawaw, Japan, 2009, pp. 103-108.
- [18] Md. M. Hasan, *A Low-Cost Realization of Quantum Ternary Adder Using Muthukrishnan-Stroud Gate*, ICECE, Dhaka, 2008, pp. 732-734.
- [19] M. H. A. Khan, M. Perkowski, *Evolutionary Algorithm Based Synthesis of Multi-Output Ternary Functions Using Quantum Cascade of Generalized Ternary Gates*, International Journal on Multiple-Valued Logic and Soft Computing, 2005.